En este capítulo seguiremos trabajando con señales digitales. En el capítulo anterior vimos cómo “escribirlas”, lo que nos permitió encender y apagar un dispositivo; ahora aprenderemos cómo “leerlas”. Nuestro principal objetivo es ser capaces de detectar si se está o no recibiendo una determinada señal para poder actuar en consecuencia. Con los conocimientos adquiridos podremos incorporar en los proyectos un elemento fundamental en cualquier dispositivo electrónico: la interacción con el usuario.

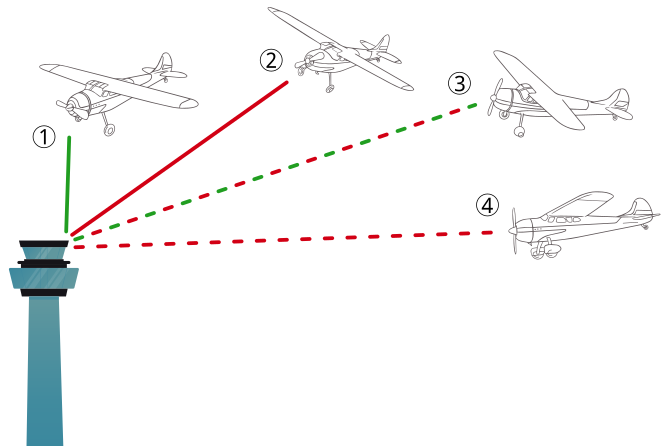
En cuanto al material, junto a la placa, la *protoboard* y algunos cables de conexión usaremos:

* Un led verde.
* Un led rojo.
* Dos resistencias de 220 Ω.
* Tres pulsadores.

6.1 Proyecto “Lámpara de señales”

La Organización de Aviación Civil Internacional es el organismo que fija las normas y recomendaciones que deben seguir los Estados en cuanto a las reglas generales de vuelo y uso del espacio aéreo. En su publicación *Reglamento del aire* se recogen las señales luminosas que, en el caso de un fallo en las comunicaciones por radio, permiten el control del tráfico aéreo. Para emitir estas señales se utiliza una lámpara de señales.

Una lámpara de señales es un dispositivo que emite un haz de luz –verde, rojo o blanco– que puede estar en estado continuo o intermitente. Un operario en la torre de control dirige hacia la aeronave este haz para informar al piloto de cómo debe proceder. Por ejemplo, si el piloto de una nave en vuelo recibe una luz verde fija está autorizado para aterrizar; una luz roja fija le indica que debe ceder el paso a las otras aeronaves; una luz roja y verde alterna es una advertencia cuyo significado es “Extreme la precaución”, y una serie de destellos rojos le avisa de que el aeródromo es peligroso y que, por tanto, no debe aterrizar.

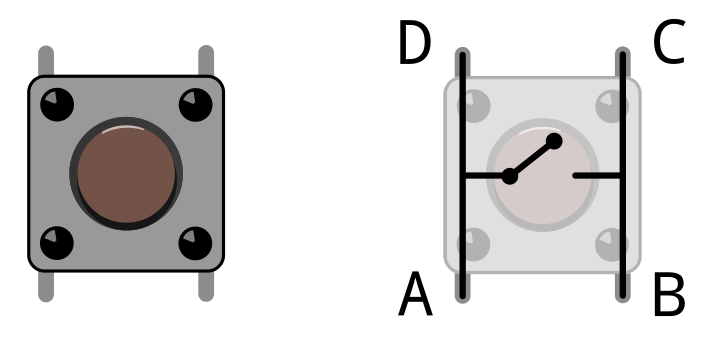


Distintas señales luminosas dirigidas desde el control de aeródromo a las aeronaves en vuelo: (1) autorizado para aterrizar, (2) ceda el paso a las otras aeronaves, (3) extreme la precaución y (4) no aterrice.

Nuestra misión en este capítulo es fabricar un prototipo de una lámpara de señales. Las señales serán emitidas por dos ledes, uno verde y uno rojo, y la selección del tipo de señal se hará mediante pulsadores.

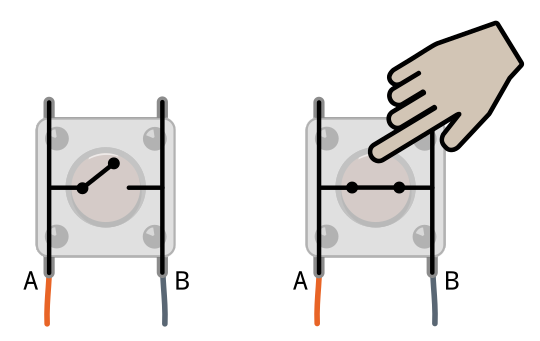
6.2 El pulsador

Un elemento fundamental en cualquier circuito eléctrico es un interruptor. Un interruptor, como su nombre indica, sirve para interrumpir el paso de la corriente eléctrica –por ejemplo, el interruptor que usamos a diario para apagar y encender la luz–. Su funcionamiento se basa en impedir o permitir la conexión eléctrica entre sus terminales. Pues bien, en electrónica es muy habitual utilizar unos interruptores momentáneos, los pulsadores, que solo permiten el paso de la corriente mientras están siendo pulsados; una vez se sueltan, la corriente se vuelve a interrumpir. A estos pulsadores se les llama “normalmente abiertos”, ya que en estado normal (es decir, no pulsado) producen que el circuito esté abierto y, en consecuencia, no circula la corriente.



Izquierda: Un pulsador. Derecha: El pulsador, por dentro.

Un pulsador habitualmente tiene cuatro patitas o terminales. Sin embargo, para conectarlo solo hacen falta dos cables. Entonces, ¿a qué terminales se conectan los cables? Para contestar esta pregunta debemos ver cómo es el pulsador por dentro. Nombrando los terminales con las letras A, B, C y D, como en el dibujo, los puntos A y D están unidos internamente y, por tanto, eléctricamente son el mismo punto; lo mismo se puede decir de B y C. En conclusión, para hacer las conexiones debemos conectar uno de los cables al terminal A o D, y el otro a B o C. De esta manera se consigue que, mientras no se pulsa el pulsador, el circuito está abierto y no circula la corriente; cuando se pulsa, el circuito se cierra y, en consecuencia, circula la corriente.



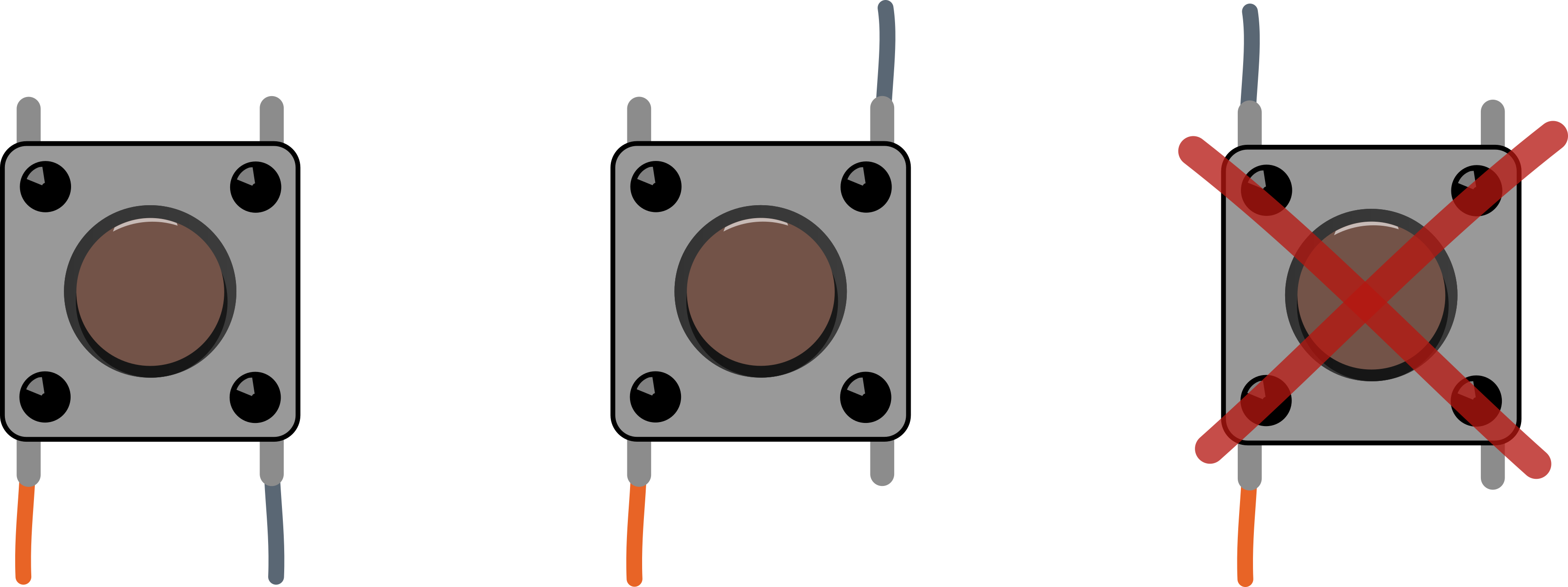
Funcionamiento de un pulsador normalmente abierto. Izquierda: Cuando el pulsador está en estado normal –no pulsado– el interruptor está abierto, por lo que los terminales A y B no están conectados y no circula la corriente. Derecha: Al pulsarlo, y solo mientras se mantiene pulsado, se cierra el interruptor y, en consecuencia, se conectan eléctricamente A y B, permitiendo el paso de la corriente.

6.3 Conexiones

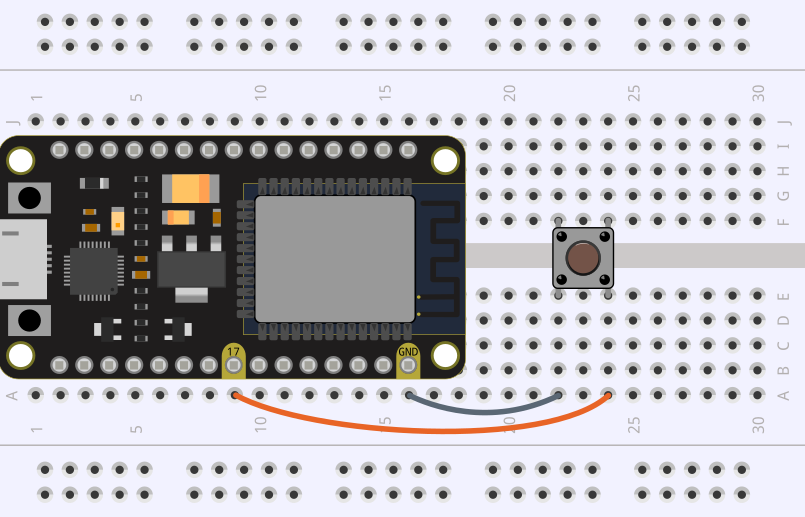
Siguiendo las indicaciones del apartado anterior, conectamos el pulsador al microcontrolador de la siguiente manera:

* Un terminal del pulsador a un pin digital del microcontrolador.
* El otro terminal del pulsador a GND.

Dado que un pulsador no tiene polaridad, es irrelevante cuál de ellos se conecte al pin de señal y cuál a GND.

Dos maneras correctas de conectar el pulsador (izquierda y centro) y una incorrecta (derecha).

En este ejemplo usaremos el GPIO17 para leer el estado del pulsador. Este GPIO tiene un nombre diferente dependiendo de la placa: en Arduino es el D8; en ESP32, dependiendo del modelo, puede ser el TX2 o el G17, y en la Raspberry Pi Pico es el pin número 22.



|  |  |  |  |
| --- | --- | --- | --- |
|  | Arduino | ESP32 | Raspberry Pi Pico |
| GPIO17 | D8 | TX2 / G17 | 22 |

¿Y la resistencia?

Esto es todo lo que se necesita para utilizar un pulsador en un proyecto, aunque con un matiz. Para que la lectura del pulsador sea correcta, es imprescindible añadir en el montaje una resistencia. Y, sin embargo, podemos conseguir que el circuito anterior funcione perfectamente sin añadir más componentes. ¿Cómo? Resulta que muchos microcontroladores –entre los que se encuentran los que estamos utilizando– tienen una resistencia interna que podemos activar con software y así evitamos añadir la resistencia externa. ¡Muy útil!

***i***

6.4 Lectura del pulsador

Una vez que está montado el circuito, veamos cómo puede el microcontrolador leer el estado del pulsador. Dado que el pulsador proporciona una señal digital (o está pulsado o no lo está, no hay más opciones), el procedimiento es análogo al seguido cuando encendimos y apagamos el led.

El estado del pulsador lo leeremos a través de un pin de la placa, por lo que volveremos a recurrir a la clase Pin del módulo machine. Como vimos en el capítulo anterior, para trabajar con las funciones de la case Pin primero hay que crear un objeto Pin, que no es más que una abstracción en lenguaje MicroPython del pin físico. Dado que ahora el pin tiene conectado un pulsador cuyo valor queremos leer –en vez de un led cuyo valor queremos escribir–, aquí es donde aparecen las primeras diferencias.

Recordemos la sintaxis del constructor Pin(), ya que ahora aparece un argumento nuevo.

|  |
| --- |
| Pin(id, mode=x, pull=x) |
| Crea una instancia de un objeto Pin. Argumentos:   * id: identificador del pin que queremos controlar. * mode: como va a ser un pin de lectura tomará el valor Pin.IN. * pull: determina si se activa o no la resistencia interna del microcontrolador. Toma el valor None si no se activa (es el valor por defecto) o el valor Pin.PULL\_UP si se activa. Dado que para simplificar el montaje no hemos incluido una resistencia externa, debemos activarla.   No es necesario especificar el nombre de los parámetros mode y pull si se mantiene el orden en el que están definidos en el constructor. |

Las siguientes instrucciones crean una instancia del objeto Pin y lo asocian a la variable PULSADOR. De esta manera, cada vez que queramos acceder al pin en el que está conectado el pulsador lo haremos a través de dicha variable.

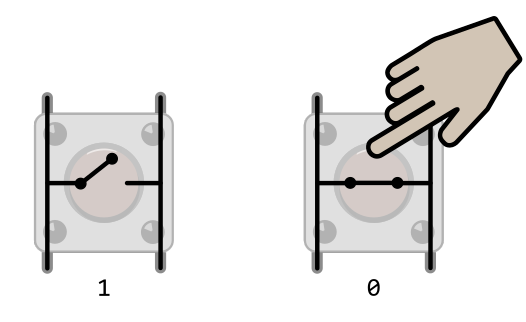
>>> from machine import Pin

>>> PULSADOR = Pin(17, Pin.IN, Pin.PULL\_UP)

Uno de los métodos de la clase Pin es value(). Vimos con anterioridad que esta función, usada sin argumentos, devuelve el valor lógico en el que se encuentra el pin. Por tanto, para leer el valor en el que está el pulsador aplicaremos dicho método al objeto PULSADOR recién creado.

>>> PULSADOR.value()

Ejecutamos la instrucción anterior en el REPL, tanto con el botón pulsado como con el botón sin pulsar. Cuando el botón está sin pulsar el valor devuelto por value() es 1, y cuando está pulsado es 0.



Con el pulsador abierto, PULSADOR.value() toma el valor 1. Cuando se pulsa, el interruptor se cierra y PULSADOR.value() vale 0.

Con este sencillo ejemplo hemos conseguido leer una señal digital con el microcontrolador.

6.5 ¿Pulsado o no?

Ahora que ya sabemos cómo funciona el pulsador, proponemos al lector que escriba un programa que muestre continuamente el estado en que se encuentra el pulsador. Para hacerlo no se necesita ningún concepto más allá de lo visto hasta ahora.

Recogemos aquí una posible solución.

from machine import Pin

from time import sleep\_ms

PULSADOR = Pin(17, Pin.IN, Pin.PULL\_UP)

while True:

print(PULSADOR.value())

sleep\_ms(200)

Algo que se debe hacer cuando se lee el estado de un pulsador es añadir un pequeño retardo antes de volver a leerlo; por este motivo hemos añadido la instrucción sleep\_ms(). Al ejecutar el programa se obtiene una retahíla de unos si el botón no está pulsado, y de ceros mientras el botón está pulsado. Lo anterior se puede mejorar imprimiendo un mensaje que sea más significativo que los ceros y unos; por ejemplo, “Pulsado” y “No pulsado”. Para ello primero debemos comprobar en cuál de los dos estados se encuentra el pulsador, para lo que utilizaremos el condicional if-else.

La instrucción if PULSADOR.value() == 1 quiere decir “si PULSADOR está en estado 1”, es decir, no pulsado. A su vez, la instrucción else implica el caso contrario, o sea, que PULSADOR esté en estado 0 o pulsado.

from machine import Pin

from time import sleep\_ms

PULSADOR = Pin(17, Pin.IN, Pin.PULL\_UP)

while True:

if PULSADOR.value() == 1:

print("No se está pulsando el botón")

else:

print("Se está pulsando el botón")

sleep\_ms(200)

Se obtiene el mismo resultado que con los ceros y los unos, pero ahora es más fácil de entender.

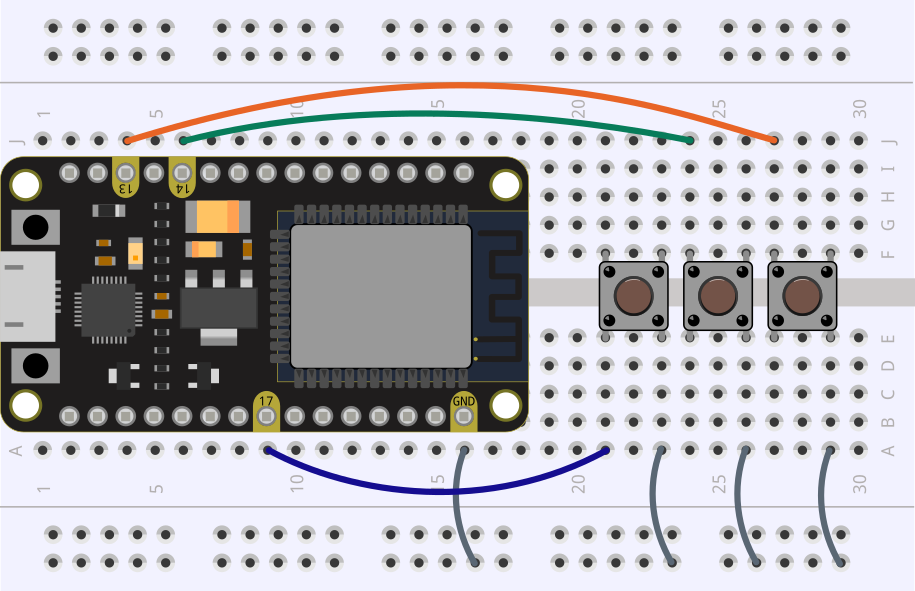
6.6 Tres pulsadores

Para controlar tres pulsadores la única complicación es que hay más cables para conectar y el programa será un poco más largo pero, por lo demás, nada nuevo.

Para montar el circuito con tres pulsadores, además del GPIO17 utilizaremos el GPIO13 y el GPIO14. En la tabla inferior mostramos los números de dichos pines en diferentes placas (aunque en la placa Arduino dos pines están marcados como analógicos, todos los pines analógicos pueden ser usados como digitales).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | GPIO | Arduino | ESP32 | Raspberry Pi Pico |
| Pulsador 1 | GPIO17 | D8 | TX2 / G17 | 22 |
| Pulsador 2 | GPIO14 | A7 | D14 / G14 | 19 |
| Pulsador 3 | GPIO13 | A6 | D13 / G13 | 17 |

Si la placa ESP32 es demasiado ancha para poder hacer las conexiones como en la imagen se pueden utilizar dos *breadboards* juntas.



Más adelante haremos cosas más sofisticadas con los tres pulsadores pero, por ahora, lo único que queremos es mostrar en el REPL cuál es el botón que se está pulsando. Esta vez usaremos el condicional if-elif-else de Python para comprobar en qué botón la señal es 0 (es decir, qué botón está pulsado).

from machine import Pin

from time import sleep\_ms

PULSADOR\_1 = Pin(17, Pin.IN, Pin.PULL\_UP)

PULSADOR\_2 = Pin(14, Pin.IN, Pin.PULL\_UP)

PULSADOR\_3 = Pin(13, Pin.IN, Pin.PULL\_UP)

while True:

if PULSADOR\_1.value() == 0:

print("Se está pulsando el botón 1")

elif PULSADOR\_2.value() == 0:

print("Se está pulsando el botón 2")

elif PULSADOR\_3.value() == 0:

print("Se está pulsando el botón 3")

else:

print("No se está pulsando ningún botón")

sleep\_ms(200)

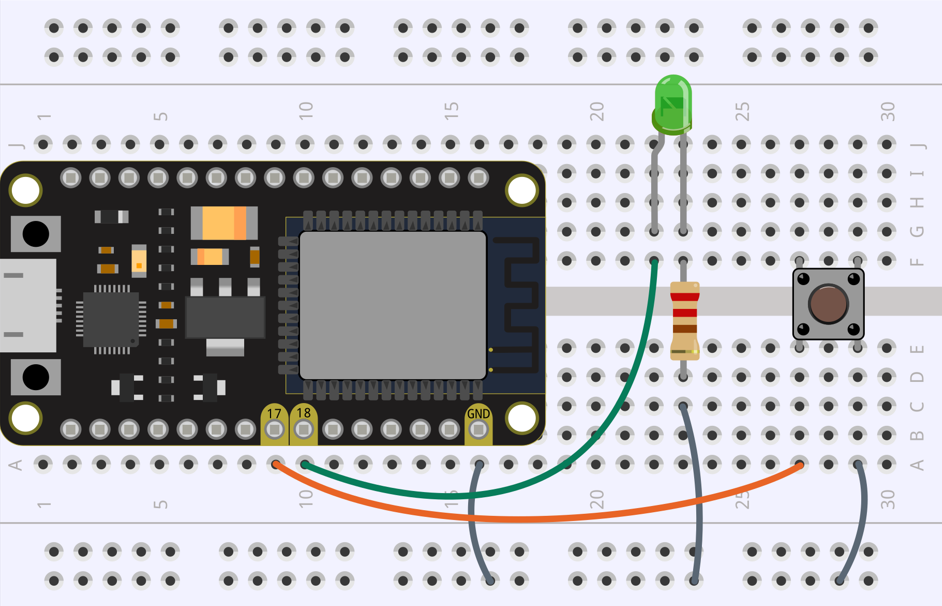
Al ejecutar el programa, dependiendo del botón que se pulse, obtendremos la salida correspondiente.

6.7 El led y el pulsador

El paso siguiente consiste en utilizar la información leída en un pulsador para hacer algo útil. ¿Qué tal si, por ejemplo, usamos el pulsador para encender y apagar un led? De nuevo, los conceptos necesarios ya han sido tratados con anterioridad, así que animamos al lector a hacer el montaje y escribir los programas por su cuenta antes de mirar la solución.

Lo primero es montar el circuito. Solo tenemos que poner juntos los dos montajes ya realizados –el del led y el del pulsador–. Usaremos los siguientes pines (pueden cambiarse):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | GPIO | Arduino | ESP32 | Raspberry Pi Pico |
| Pulsador | GPIO17 | D8 | TX2 / G17 | 22 |
| Led | GPIO18 | D9 | D18 / G18 | 24 |



Hay diferentes maneras de controlar un led usando un pulsador. Una manera consiste en que el led se mantenga encendido únicamente mientras el botón está siendo pulsado, y se apague cuando se deja de pulsar. También se puede hacer que el estado del led cambie de encendido a apagado, y viceversa, cada vez que se pulse el botón. Veremos ambas opciones en los próximos apartados.

6.8 Manejando el led con el pulsador (versión 1)

Una manera de manejar el led es que se encienda únicamente mientras el botón esté pulsado y se apague cuando se deja de pulsar. Para conseguirlo comprobamos si el valor de PULSADOR es 0, lo que indica que el botón está pulsado; en ese caso se enciende el led. Si, por contra, el valor de PULSADOR no es 0, el botón no está siendo pulsado y el led se apaga. Al poner esta comprobación dentro del bucle while True el microcontrolador está continuamente controlando el estado del pulsador para decidir si encender o apagar el led.

from machine import Pin

PULSADOR = Pin(17, Pin.IN, Pin.PULL\_UP)

LED = Pin(18, Pin.OUT)

while True:

# Botón pulsado

if PULSADOR.value() == 0:

LED.value(1)

# Botón no pulsado

else:

LED.value(0)

6.9 Manejando el led con el pulsador (versión 2)

En esta ocasión deseamos que el estado del led cambie de encendido a apagado, y viceversa, cada vez que se pulsa el botón. En tal caso, la pulsación del botón (PULSADOR.value() == 0) produce el cambio en el estado actual del led: si el led estaba en estado 0 pasa a estado 1, y si estaba en estado 1 pasa a 0. De este cambio en el estado se encarga el operador not.

from machine import Pin

from time import sleep\_ms

PULSADOR = Pin(17, Pin.IN, Pin.PULL\_UP)

LED = Pin(18, Pin.OUT, value=0)

while True:

if PULSADOR.value() == 0:

LED.value(not LED.value())

sleep\_ms(200)

6.10 Lámpara de señales

Estamos ya en condiciones de afrontar el reto propuesto al principio del capítulo: construir una lámpara de señales. Como en todos los circuitos y programas presentados en este manual, siempre hay más de una manera de hacer las cosas. Aquí presentamos una posible solución para que sirva como ayuda. Pero seguro que es mejorable, así que se puede modificar, quitar o ampliar lo que se desee para darle un toque personal al proyecto.

El problema

Empezaremos definiendo el problema que queremos resolver. Para que el programa no quede demasiado largo, nuestra lámpara de señales solo va a emitir tres señales:

* Luz verde fija, que indica autorización para aterrizar.
* Serie de destellos rojos, para avisar de que el aeródromo es peligroso y la aeronave no debe aterrizar.
* Luz alterna roja y verde, señal de advertencia para extremar la precaución.

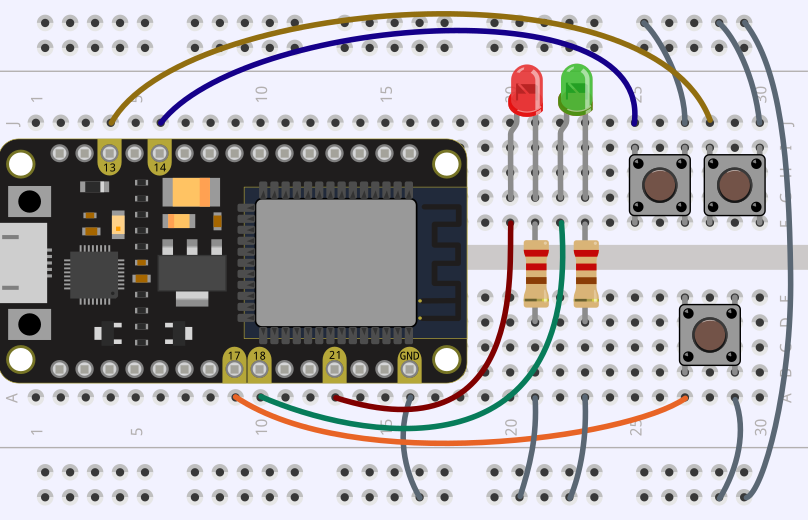
Cada señal será activada por un pulsador diferente. Además, la señal solo estará activa mientras se esté pulsando el botón correspondiente.

|  |  |  |
| --- | --- | --- |
| Pulsador | Señal | Significado |
| 1 | Verde fija | Autorizado para aterrizar |
| 2 | Serie de destellos rojos | Peligro, no aterrice |
| 3 | Alterna roja y verde | Extreme la precaución |

El circuito

Como hemos establecido en las especificaciones del problema, utilizaremos tres pulsadores (uno para cada señal) y dos ledes, uno verde y otro rojo. Los pines de conexión que hemos elegido son los siguientes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | GPIO | Arduino | ESP32 | Raspberry Pi Pico |
| Led verde | 18 | D9 | D18 / G18 | 24 |
| Led rojo | 21 | D10 | D21 / G21 | 27 |
| Pulsador 1 | 17 | D8 | TX2 / G17 | 22 |
| Pulsador 2 | 14 | A7 | D14 / G14 | 19 |
| Pulsador 3 | 13 | A6 | D13 / G13 | 17 |



El programa

from machine import Pin

from time import sleep\_ms

PULSADOR\_1 = Pin(17, Pin.IN, Pin.PULL\_UP)

PULSADOR\_2 = Pin(14, Pin.IN, Pin.PULL\_UP)

PULSADOR\_3 = Pin(13, Pin.IN, Pin.PULL\_UP)

LED\_VERDE = Pin(18, Pin.OUT, value=0)

LED\_ROJO = Pin(21, Pin.OUT, value=0)

# Tiempo entre destellos (en ms)

INTERVALO = 500

def apagado():

LED\_VERDE.value(0)

LED\_ROJO.value(0)

def aterrice():

LED\_VERDE.value(1)

def peligro():

LED\_ROJO.value(not LED\_ROJO.value())

sleep\_ms(INTERVALO)

def precaucion():

LED\_VERDE.value(1)

LED\_ROJO.value(0)

sleep\_ms(INTERVALO)

LED\_VERDE.value(0)

LED\_ROJO.value(1)

sleep\_ms(INTERVALO)

while True:

if PULSADOR\_1.value() == 1 and PULSADOR\_2.value() == 1 and PULSADOR\_3.value() == 1:

apagado()

elif PULSADOR\_1.value() == 0:

aterrice()

elif PULSADOR\_2.value() == 0:

peligro()

elif PULSADOR\_3.value() == 0:

precaucion()

Hemos utilizado el operador lógico and de Python para comprobar si varias condiciones se verifican al mismo tiempo (que los tres pulsadores estén en estado no pulsado). Ejecutando el programa comprobamos que se emite la señal correspondiente cada vez que se pulsa un botón, y que si no hay ningún botón pulsado ambos ledes están apagados.

Subirlo a la placa

Ya tenemos montada la lámpara de señales y hemos comprobado que funciona correctamente. Si guardamos el programa en la placa con el nombre main.py y la conectamos a un *power bank* el programa se ejecutará desde el microcontrolador, con lo que ya tenemos un dispositivo independiente del ordenador.

En resumen

El presente capítulo complementa lo visto en el anterior sobre señales digitales. Hemos aprendido cómo se lee una señal digital; en particular, hemos accedido al valor del estado de un pulsador. Para ello hemos introducido un nuevo argumento, pull, en el constructor Pin(). Hemos utilizado la estructura condicional if-elif-else para controlar el flujo de ejecución de un programa y hemos recurrido al uso de funciones para hacer más modular el código. Finalmente hemos juntado los conocimientos sobre lectura y escritura de señales digitales aplicándolos al caso práctico de controlar un led usando un pulsador. Con todo lo aprendido hemos montado un sencillo sistema de control de guiado de aeronaves con luces led.